# SkOwl 1.5

## - Unpublished Technical Report -

Michael D. Kickmeier-Rust
*Cognitive Science Section, Department of Psychology*
*University of Graz, Austria*

## SkOwl at a glance

**SkOwl** is a software tool offering semi-automatic support of transforming OWL domain ontologies, concept maps, or proposition files to the format of the ELEKTRA ontology model (Kickmeier-Rust & Albert, in press). Moreover, **SkOwl** supports and easy definition of skills on the basis of propositions and action verbs and it allows semi-automatically generating prerequisite relations between skills on the basis of the set inclusion principle, action verbs, and manual work-ups.

**SkOwl …**

- reads OWL ontologies, concept maps, proposition files and extracts the available propositions
- supports the semiautomatic and user-friendly definition of skills on the basis of propositions
- includes the supports the semiautomatic and user-friendly assignment of action verbs to skills
- supports a semiautomatic and most user-friendly generation of prerequisite relations among skills
- supports the manipulation of prerequisite relations between skills on a visual basis
- enables drawing prerequisite relations and even skill/knowledge structures
- transformation of skills and prerequisite relations to the format of the ELEKTRA ontology model
- provides a variety of export formats

**SkOwl 1.5**

Copyright © 2007 Cognitive Science Section, Department of Psychology, University of Graz

Author: Michael D. Kickmeier-Rust

**Vector Graphics ActiveX**

Copyright © 2006 S&G Team

*The software and its source code (SkOwl 1.5) described in this technical report are distributed as freeware under the GNU General Public License provided by the Free Software Foundation. This form of distribution license does not include the distribution of Vector Graphics ActiveX. This third-party product is distributed under separate conditions. For detailed terms and conditions of use and (re-)distribution see section 4.*

The brand names Windows, Protégé, Java, Acrobat, Acrobat Reader, Internet Explorer, SQL Server, Access, cmap tools, and other brand names occurring in this technical report are registered trademarks of the respective owners in Europe, in the United States, and other countries.

**Table of contents**


Section 1: Purpose and scientific background ..... 5
1.1. Purpose and area of application ..... 5
1.2. Ontologies ..... 5
1.3. The ELEKTRA ontology model ..... 7
1.4. Concept maps and skill definition ..... 8
1.4.1. Defining skills by substructure set-inclusion ..... 9
1.4.2. Adding the action component: Using the component-attribute approach ..... 10
Section 2: Software manual ..... 12
2.1 SkOwl's main application window ..... 12
2.1.1. Opening source files ..... 12
2.1.2. Defining skills – the Skills tab ..... 17
2.1.2.1. Propositions ..... 17
2.1.2.2. Skill descriptions ..... 17
2.1.2.3. Action verbs ..... 18
2.1.2.4. Add a skill manually ..... 22
2.1.2.5. Combining skills from multiple files ..... 22
2.1.2.6. Add a proposition manually ..... 22
2.1.2.7. Changing node colors ..... 22
2.1.2.8. Other functions ..... 23
2.1.3. Saving and exporting ..... 23
2.1.4. Generating a basis / prerequisite relation ..... 24
2.1.5. Print skill list ..... 26
2.1.6. Editing action verb favorites catalogue ..... 27
2.1.7. Application options ..... 27
2.1.8. Help and additional function ..... 28
2.2 SkOwl's relation visualizer ..... 29
2.2.1 Starting the relation visualizer module ..... 30
2.2.2 Open source files ..... 30
2.2.3 Saving basis files and images ..... 31
2.2.4 Printing images ..... 32
2.2.5 Manipulating the graph ..... 32
2.2.5.1 Changing colors ..... 33
2.2.5.2 Changing font style ..... 33
2.2.5.3 Changing pen (line) width ..... 33
2.2.5.4 Zooming in and out ..... 33
Section 3: Architecture and Specifications ..... 34
3.1. Architecture ..... 34
3.2. System Requirements ..... 36
3.3. Action verb dictionary ..... 37
3.4. Error codes ..... 38
Section 4: License agreements ..... 43
4.1. SkOwl ..... 43
4.2. Vector Graphics Active X ..... 49
Section 5: References ..... 51

# Section 1: Purpose and scientific background

## 1.1. Purpose and area of application

**SkOwl** is a software tool offering semi-automatic support of transforming OWL domain ontologies, concept maps, or proposition files to the format of the ELEKTRA ontology model. Moreover, **SkOwl** supports and easy definition of skills on the basis of propositions and action verbs and it allows semi-automatically generating prerequisite relations between skills on the basis of the set inclusion principle, action verbs, and manual work-ups.

## 1.2. Ontologies

Originally, the term ontology was established in philosophy where it describes a discipline dealing with existence. The term was introduced to computer science by Tom Gruber (1993) where it describes “a formal, explicit specification of a shared conceptualization”.
An ontology provides a structured and semantically rich approach to model a certain domain; it is a data model representing entities of a domain and the relations among them. Ontologies count classes, individuals, inheritances, and relationships between classes as their major components. The entities of an ontology can be described with attributes, each having a name, a certain data type, and one or more values (e.g., attribute ”language“ and value “English”). Attributes, moreover, serve to establish relations among entities. For example, a relation might be “A is a subclass of B”.

In the context of e-Learning ontologies may serve as a means of achieving semantic precision between a domain of learning material and the learner’s prior knowledge and learning goals. Ontologies bridge the semantic gap between humans and machines and, consequently, they facilitate the establishment of the semantic web and build the basis for the exchange and re-use of contents that reaches across people and applications (e.g., web-services). The main requirements to ontologies are a well-defined syntax, well-defined semantics, efficient reasoning support, sufficient expressive power, and convenience of expression (Antoniou & van Harmelen, 2004).

From a technical perspective, an ontology is a text-based reference of information, represented using the syntax of an *ontology representation language* (Gómez-Pérez & Corcho, 2002). Most of them are built upon XML and RDF. There is a variety of such representation languages (see Antoniou & van Harmelen, 2004 for a review). One of the most popular are OIL (Ontology Inference Layer) and DAML+OIL. An ontology that is developed in any such representation language is usually converted into an RDF/XML-like form and can be -at least partially- parsed even by common RDF/XML parsers.

To extract information from ontologies, reasoners -especially for OWL- are available, e.g., RACER (http://www.sts.tu-harburg.de/~r.f.moeller/racer/) or Pellet (http:// pellet.owldl.com/). Reasoning might refer to class memberships (is *i* an instance of class *C*?), to the equivalence of classes (is *C* equivalent to *P*?), to consistency (if i is an instance of class *A* and *A* is a subclass of $B \cap C$ and *A* is a subclass of *D* and *B* and *D* are disjoint. *A* should be an empty set and thus there is an inconsistency, an error in the ontology, because *A* has the instance *i*). Finally reasoning might refer to classification (on the properties of an instance the related class can be identified.).

In 2004, W3C has officially released *OWL* (Web Ontology Language) as W3C recommendation for representing ontologies. OWL is developed starting from description logic and DAML+OIL. The increasing popularity of OWL might lead to its widest adoption as the standard ontology representation language on the Semantic Web in the future. Essentially, OWL is a set of XML elements and attributes, with well-defined meaning, that are used to define terms and their relationships (e.g., `Class`, `equivalentProperty`, `intersectionOf`, `unionOf`, etc.). OWL elements extend the set of RDF and RDFS elements, and the *owl* namespace is used to denote OWL encoding. OWL provides a set of constructors that allow deriving classes from other classes and a set of axioms that allow asserting subsumption or equivalence in terms of classes, individuals, or properties, the disjoitness of classes, or properties of properties.

OWL consists of three species, the full language (OWL Full) which includes the entire OWL language and which is fully upward compatible with RDF both systematically as well as semantically. The disadvantage however is that it is so powerful that complete reasoning support is difficult. OWL DL (DL is stands for description logic) basically disables the application of OWL's constructors (e.g., `oneOf`) to each other. This allows a more efficient support of reasoning. Third, OWL *Lite* which is a further restricted variant. *Lite* classes can only be defined in terms of named superclasses (superclasses cannot be arbitrary expressions), and only certain kinds of class restrictions can be used. Equivalence between classes and

subclass relationships between classes are also only allowed between named classes, and not between arbitrary class expressions. Similarly, restrictions in OWL *Lite* use only named classes. OWL *Lite* also has a limited notion of cardinality - the only cardinalities allowed to be explicitly stated are 0 or 1.

## 1.3. The ELEKTRA ontology model

Many of today's ontologies or semantic web approaches to educational technology focus on an ontology model that is centered on domain concepts and learning objects (e.g., Branko Neto & Gauthier, 2006 or Mizoguchi & Bourdeau, 2000) and often ontologies of pedagogical model, learner model, and domain are separated (e.g., Sampson, Karagiannidis, & Cardinali, 2002).

The ELEKTRA ontology model is supposed to address the requirements of providing adaptive interventions in the game in order to balance challenge and ability and therefore not only providing successful learning paths within the game's narrative but also to retain motivation and even flow experience. Thus, the ontology model incorporates the concepts related to adaptive interventions on a macro as well as on a micro-level, that is, it models problem solution spaces for problem solving tasks within the game environment. Although the re-usability of learning objects is more difficult in digital games for educational purposes because the learning objects are an integral part of the entire game which is not interchangeable between different games in most cases, the presented ontology model is supposed to serve the growing demands on standardization and semantically rich resource description in the context of educational technologies also. As mentioned before, some authors argue that currently a lack of commonly accepted resource description standards for learning objects exists. One reason might be a focus on learning objects in current approaches. In the presented ontology model we introduce a focus on the learner and, therefore, on latent skills. This approach might be a more comprehensive and easier to standardize method for describing and defining LOs (learning objects) and learning objectives. While learning objects are most often strongly interlaced with instructional methods or events, the focus on underlying skills offers a cornerstone, which is not only directly related to human abilities and learning objectives but which also allows a very precise description of Los.

From a technical perspective the ELEKTRA ontology model builds upon OWL DL; due to its popularity it is the quasi-standard for ontology representation languages. All classes have

the RDF-attributes label and description. The ontology model is illustrated in Fig. 1.1. Most relations between the classes' instances are non-functional (marked with an asterisk in Fig. 1.1), meaning that one or more instances of one class can be associated with the instances of a related class. For further details refer to Kickmeier-Rust & Albert (in press).

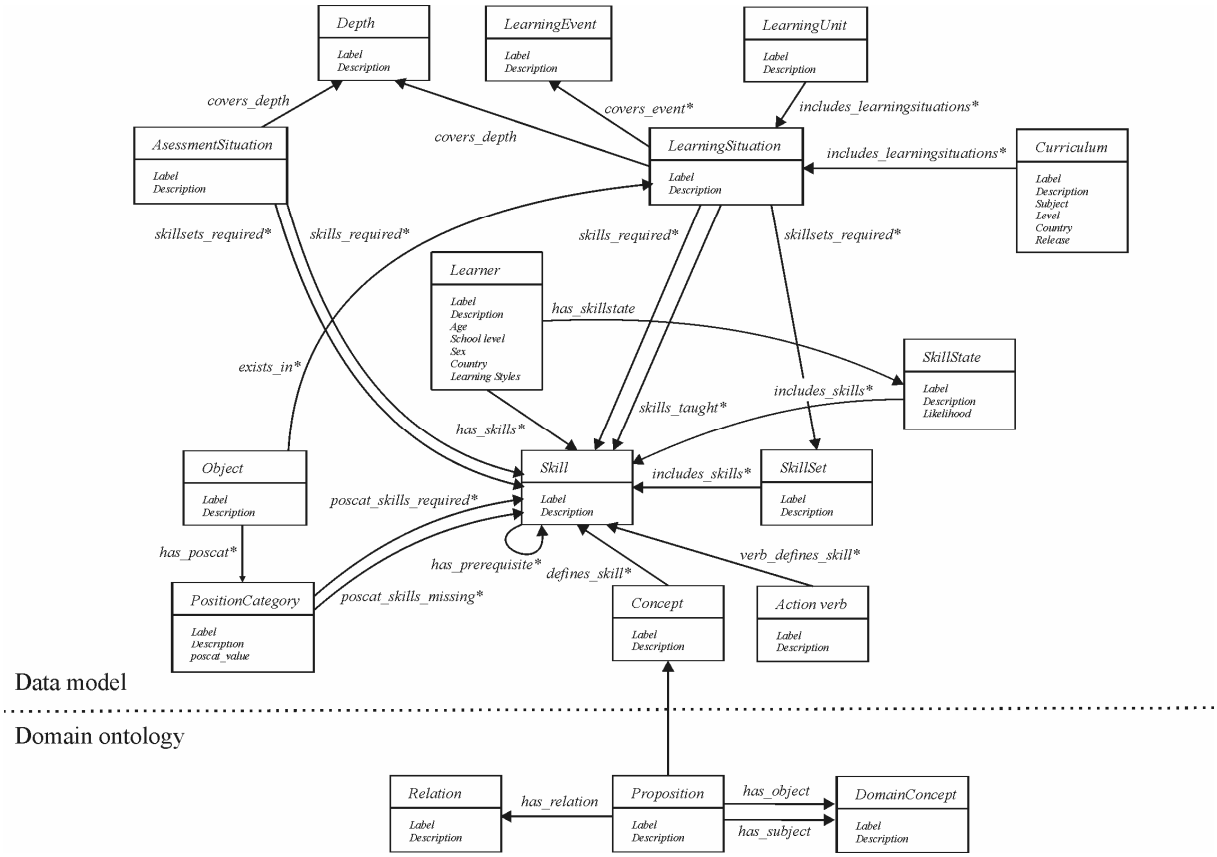


*Figure 1.1.* The ELEKTRA ontology model

## 1.4. Concept maps and skill definition

Skills represent abilities and cognitive constructs underlying the observable solution or learning behavior in a particular knowledge domain. They can be described in the tradition of Anderson (e.g. Anderson, 1996) by distinguishing two components, a declarative and a procedural one. The declarative part refers to conceptual (semantic) knowledge about facts, which is represented in a concept map. The procedural part, on the other hand, refers to

actions aiming at applying the declarative knowledge in a problem solving or learning context. When using a skill, the declarative (conceptual) and the procedural (action) components are related and tuned into their application. For further details refer to Steiner & Albert (in press).

### 1.4.1. Defining skills by substructure set-inclusion

This approach builds upon a concept map providing a fine-grained level of description of the respective knowledge domain. Each skill is associated with those propositions of a concept map that correspond to the declarative component of the respective skill. Dependencies among skills can then be deduced by proceeding in the same way as described before, i.e. by applying the rule of set inclusion. This means, if the representation of a skill *S* on the concept map is a subset of that of skill *T*, then *S* can be understood as a prerequisite for *T*.

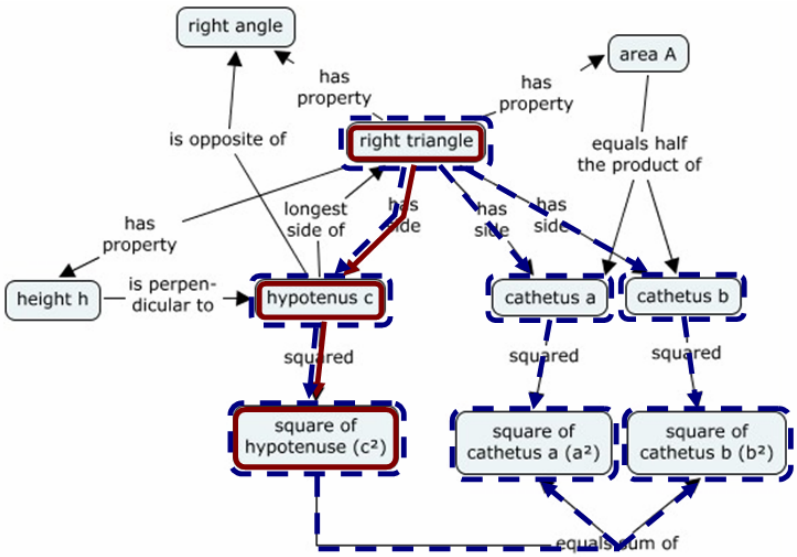


*Figure 1.2.* Concept map covering the domain of right triangles

For a simple example consider the two skills '*calculate the square of the hypotenuse*' and '*apply the Pythagorean Theorem*' from the knowledge domain of right triangles. The substructures of the concept map corresponding to these two skills are depicted in Fig. 1.2. As can be seen, the propositions representing the first skill constitute a subset of the propositions representing the second skill, therefore it can be assumed that 'calculate the square of the hypotenuse' is a prerequisite to 'apply the Pythagorean Theorem'. Consequently, having

determined prerequisite relationships among a set of skills, a competence structure can be established in analogy to the procedure of building a knowledge structure. The respective competence structure can then be utilized for creating a personalized learning path by selecting and presenting learning objects correspondingly. This means, for example, that first a learning object with the underlying skill of calculating the square of the hypotenuse will be presented, before providing the learner with learning objects that teach the skill of applying the Pythagorean Theorem.

### 1.4.2. Adding the action component: Using the component-attribute approach

The second approach for building a competence structure utilizes a concept map on a higher level of abstraction, i.e. providing a more general description of a knowledge domain. Such a concept map may for example result from curriculum or content analysis, which aims at uncovering and representing the basic concepts of a certain topic or subject and prerequisite relationships among them - for instance corresponding to the recommended or practiced sequence of teaching. In case of such a more abstract concept map, skills can in general be associated with one concept – in contrast to a subset of concepts and relations as in the previously described procedures. The special feature of the present approach is that it also takes into account the procedural component of skills, given by an action that is related to and applies the respective declarative knowledge (concept).

For building a prerequisite relation both components, i.e. declarative and procedural, are seen as separate dimensions of skills that can feature different values or attributes. This means, the declarative component of a skill can vary with respect to the concept it refers to, and the procedural component varies with respect to the involved action. Correspondingly, a skill in this context is defined as a pair consisting of an action and a concept (e.g., Heller, Steiner, Hockemeyer, & Albert, 2006); e.g. the skill ‘apply the Pythagorean Theorem’ consists of the concept ‘Pythagorean Theorem’ and the action ‘apply’. Based on this skill definition, a prerequisite relation can be established by the use of the so-called component-attribute approach (e.g., Albert & Held, 1999). For each component a structure is established by identifying dependencies among the set of attributes (concepts or actions) that these components can take on. With respect to the concepts, this structure is derived from the concept map which represents prerequisite relationships among the concepts that can straightforwardly be adopted. For instance, the concept map may reveal for the concepts ‘Pythagorean Theorem’ ($c_1$), ‘Altitude Theorem’ ($c_2$), and ‘Euclidean Theorem’ ($c_3$) that the

first theorem is a prerequisite for the other ones, e.g. in accordance with the curriculum or the instructional sequence of text books. Additionally, a structure may be introduced on the set of actions building the different attributes of the procedural component of the skills in question. To this end, learning activity taxonomies such as Bloom's taxonomy (Bloom, 1956) or its revision by Anderson and Krathwohl (2001) serve as a valuable tool. These taxonomies define cumulative levels of cognitive processing according to which the actions can be structured. The actions 'state' ($a_1$), 'explain' ($a_2$), and 'apply' ($a_3$), for example, can be associated with the categories 'Remember', 'Understand' and 'Apply' of Bloom's revised taxonomy, which – in this order – refer to gradual increasing levels of cognitive processes.

By forming the direct product of the two components, all possible skills can be defined - each of which consisting of an action and a concept, e.g. 'state the Pythagorean Theorem' ($a_1c_1$), 'explain the Pythagorean Theorem' ($a_2c_1$) and so forth. The prerequisite relation on the set of skills is derived from the order defined on the attributes within each component.. As can be seen, for the skill 'state the Altitude Theorem' ($a_1c_2$) (or, equivalently, for $a_1c_3$), the skill 'state the Pythagorean Theorem' is a prerequisite, whereas for skill 'explain the Pythagorean Theorem' ($a_2c_1$) the skills of stating all three theorems, i.e. skills $a_1c_1$, $a_1c_2$, and $a_1c_3$, are prerequisites. The established prerequisite relation gives rise to the competence structure for those combinations of actions and concepts that can actually occur or are considered. For example that with the skill 'explain the Euclidean Theorem' ($a_2c_3$) each competence state has to contain also the skills $a_2c_1$, $a_1c_2$, $a_1c_3$, and $a_1c_1$.

# Section 2: Software manual

**SkOwl** is primarily based on two modules, the main application window (Fig. 2.3 and Fig. 2.4). The main application window is intended to define skills by propositions and action verbs and to semi-automatically establish prerequisite relations between the skills. The relation visualizer (Fig. 2.10) is a module that allows drawing the basis (prerequisite relations between skills) and revising it on a visual basis. In addition, **SkOwl** enables drawing even knowledge/skill structures.

## 2.1 SkOwl's main application window

The main application window consists of a toolbar (Fig. 2.1) allowing accessing all relevant functions and two tab pages, the OWL source tab (Fig. 2.3) and the Skills tab (Fig. 2.4). In the following, all functions and properties of the main application window are described. The description order basically follows typical working processes.

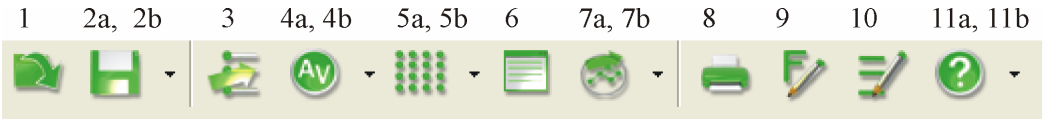


*Figure 2.1.* Toolbar of the main application window

### 2.1.1. Opening source files

To open resources click the *Open* button in the main application's toolbar (Fig. 2.1 - 1). A dialog window appears enabling the selection of a resource type to open (Fig. 2.2). **SkOwl** supports three different source types: **SkOwl** projects, OWL ontologies, and proposition lists (e.g., concept maps). To select a specific resource type, check the corresponding field and click *OK*. When opening a proposition list you have to specify a delimiter also. After clicking OK, an open file dialog window appears enabling the selection of the file.

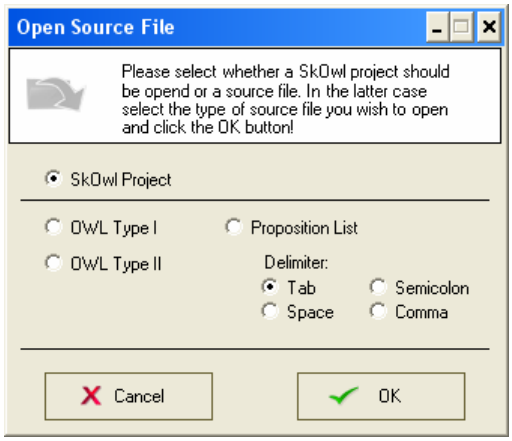


*Figure 2.2.* Open source file dialog window

- **SkOwl project**

  When opening a **SkOwl** project, the application validates the source file and opens the project. **SkOwl** automatically opens the *Skills* tab (Fig. 2.4) and displays the available propositions and the skills tree.

- **OWL type I**

  OWL type I files are well-defined domain ontologies, modeling concepts and their interrelations in form of propositions. Type I ontologies consist of four classes: A class containing the propositions and classes for subject, object, and relation. As an example, an ontology may include the classes *Propositions*, *Relations*, *Objects*, and *Subjects*. The class *Propositions* is connected to the three other classes by three relations, for example, *has_relation*, *has_object*, and *has_subject*. For objects and subjects the same class may be used, for example, the class *Concepts*. Propositions are defined by one object, one subject, and one relation (or by two concepts and one relation). By this means, a proposition might be [*House*] [*is built of*] [*Bricks*]. The advantages of type I ontologies are that they provide a well-defined internal structure and that they allow very efficient parsing, querying, and reasoning. The naming of classes does not affect the functionality of **SkOwl**!

  When opening a type I OWL file, the ontology is parsed and **SkOwl** automatically switches to the *OWL source* tab (Fig. 2.3). On this tab page the OWL source is listed in the top text area. In the class tree at the bottom, the existing classes are listed, including their relational properties (i.e., the relation each class has to other

classes) as well as the target class of each relation. The classes and the corresponding relations are visible nodes in the tree, the relations' target are hidden nodes and can be viewed by expanding the tree node. Different types are marked with different icons.

This tree representation offers a highly effective and intuitive method to display the classes and relations of ontologies, particularly large ones. In contrast to Java-based applications (e.g., Protégé), **SkOwl** uses the *Microsoft Windows treeview control* allowing to handle and display even very large data sets very quickly.

When selecting the option to open type I ontologies, make sure to select a valid type I OWL file. The current and previous versions of **SkOwl** do not check explicitly for the file's validity or type I compliance! When opening an invalid file, the source is displayed anyway and depending on the opened file, classes and relations might be displayed also. However, the extraction of valid propositions is likely compromised.

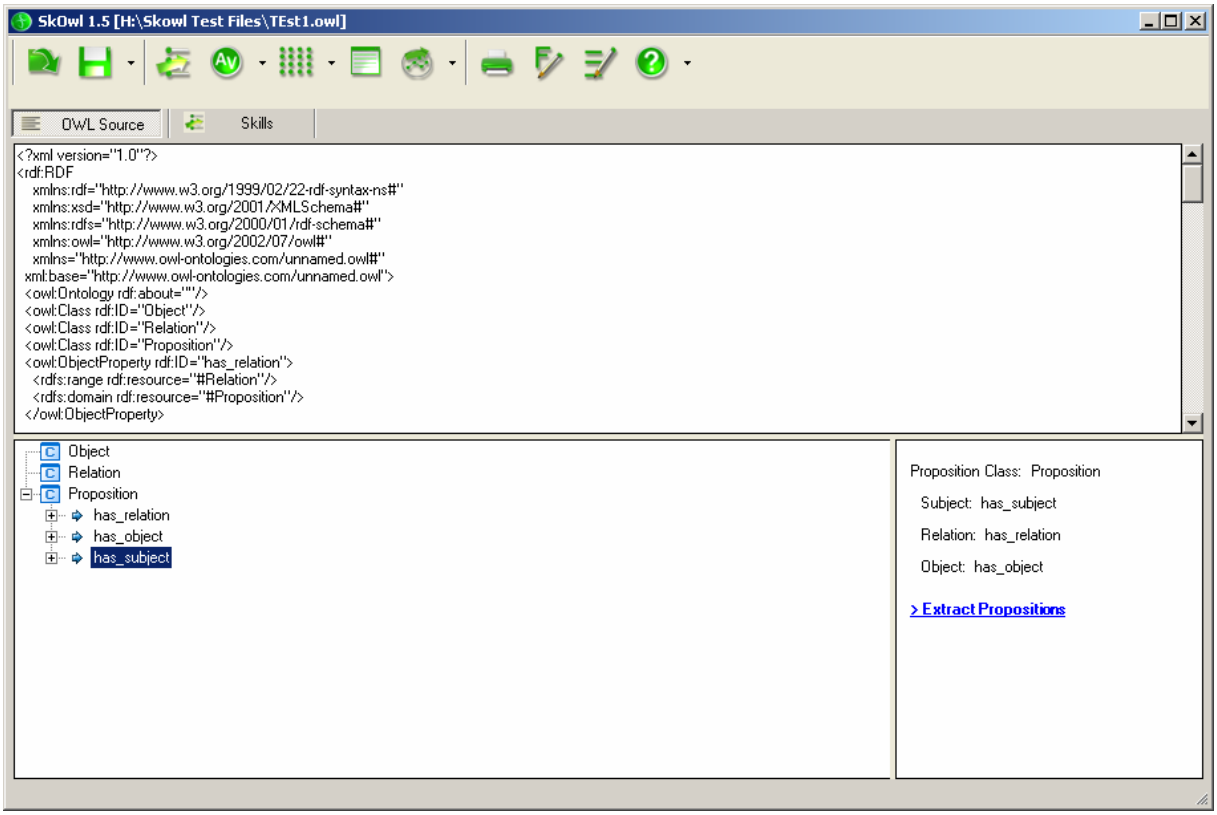


*Figure 3.3.* **SkOwl**'s *OWL source* tab

In order to extract the available propositions, in a next step, the different classes (for propositions, objects, subjects, and relations) must be identified. This is necessary

because **SkOwl** provides a generic approach, meaning that it is not sensitive to the naming of the ontology's classes.

To identify the required classes, select a class from the class tree by clicking with the left mouse button a class. Then use the right mouse button to display the context menu for each class. This menu includes following functions:

| | |
|---|---|
| *Set as Proposition Class* | Identifies the selected class as that defining the ontologies propositions |
| *Set as Subject* | Identifies the selected relation as part of the proposition class as that defining subjects (this may be the same class as that defining objects) |
| *Set as Object* | Identifies the selected relation as part of the proposition class as that defining objects (this may be the same class as that defining subjects) |
| *Set as Relation* | Identifies the selected relation as part of the proposition class as that defining all relations |
| *Expand all Nodes* | Allows displaying all tree nodes by a single mouse click |
| *Collapse all Nodes* | Allows hiding all tree nodes (with the exception of classes) with a single mouse click |

Unless all four classes have been identified, **SkOwl** cannot proceed extracting the ontology's propositions. Please make sure to identify the correct classes and relations because **SkOwl** does not provide further checks of validity. The identified classes and relations are displayed in the properties area at the bottom right. Please note that when specifying incorrect classes or relations, incorrect propositions may be completely extracted and no warning message occurs.

To start reading all propositions from the ontology click the link *Extract Propositions* in the properties area at the bottom right of the *OWL source* tab (Fig. 2.3). **SkOwl** automatically changes to the *Skills* tab. All propositions that can be parsed successfully are listed in the proposition list at the top of the *Skills* tab (Fig. 2.4).

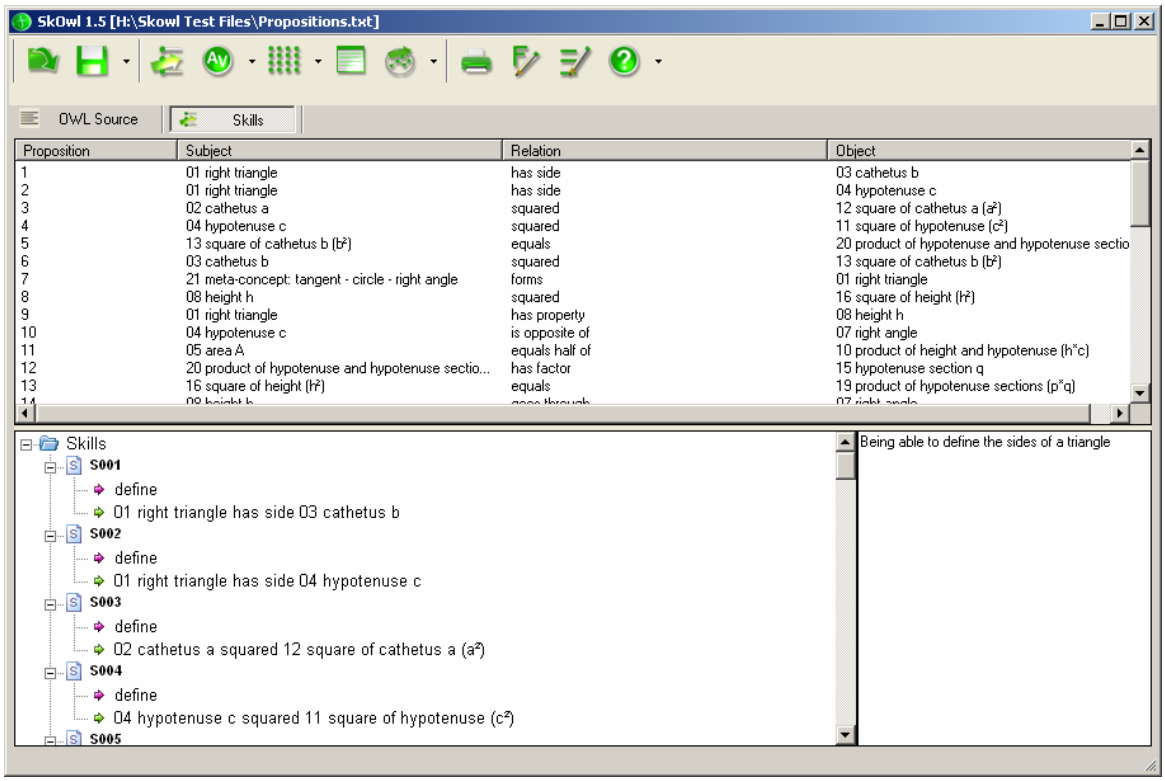


*Figure 2.4.* **SkOwl**'s Skills tab

- **OWL type II**

  OWL type II files include any other type of OWL ontologies. Please note that the current and previous versions of **SkOwl** do not include full reasoning or querying services and therefore type II ontologies may be parsed incompletely or incorrect. When opening a type II OWL file, **SkOwl** attempts to parse the ontology and then automatically switches to the *Skills* tab (Fig. 2.4), displaying the extracted propositions.

- **Proposition files**

  Proposition files are ASCII files including the proposition's name or number, subject, relations, and objects (in this order) in a single line, divided by a specific delimiter (i.e., tab, space, semicolon, or comma). Please specify the correct delimiter for your proposition file in the open source dialog window (Fig. 2.2). Please make sure to select the correct delimiter otherwise the extracted propositions may be incorrect or reading may result in an error.

  This function enables importing any type of proposition files and also concept maps, for example generated with *cmap tools*. **SkOwl** attempts to read the specified

proposition file and then automatically switches to the *Skills* tab (Fig. 2.4), displaying the extracted propositions.

### 2.1.2. Defining skills – the *Skills* tab

When a valid source file has been opened, **SkOwl** switches to the *Skills* tab (Fig. 2.4), displaying the available propositions (in the top area). This listing is ordered by the proposition's name or number, the subject, the relation, and the object.

To define skills, click the *Extract skills* button in the main toolbar (Fig. 2.1 - 3). By default, **SkOwl** defines one skill for each of the propositions and lists the skills in the skill tree (at the bottom of the *Skills* tab). For each skill a node is displayed containing, by default, a node for the corresponding proposition (Fig. 2.4). The format of skill names (by default S001 to S00n) can be defined in **SkOwl**'s options dialog (see 2.1.7).

#### *2.1.2.1.Propositions*

Most likely, skills are defined by more than a single proposition. In order to manually refine the default skill list, **SkOwl** offers easy-to-use drag and drop functionalities.

- Drag a proposition from the proposition list at the top into a skill of the skill tree. The selected proposition is automatically **added** to the corresponding skill as another node.
- Drag proposition nodes from one skill to another within the skill tree. The selected proposition / node is **moved** to the target skill.
- Drag a proposition node from one skill to another within the skill tree, holding the CNTRL key. The selected proposition is automatically **copied** to the corresponding skill as another node.
- Add a proposition **manually** selecting *Add proposition* from the skill tree's context menu. A dialog window appears allowing you to specify the propositions subject, object, and relation.

#### *2.1.2.2.Skill descriptions*

For each skill a description can be entered optionally. This description is used for the export to the ELEKTRA ontology format. To enter a skill description select a skill in the skill tree by a single left click. The text area to the right of the skill tree is enabled. Simply enter the

description and press *Enter*. The description is assigned to the selected skill and the text color turns blue. Please note that in the current version of **SkOwl** the number of characters, including spaces, is limited to 120.

*2.1.2.3.Action verbs*

In addition to propositions, skills may be defined by actions verbs. **SkOwl** offers three methods to easily apply action verbs to the available skills.

- *Apply action verbs by rules*

  **SkOwl** offers to specify rules for the application of action verbs. Open the dialog window by clicking the *Apply action verbs* button in the main toolbar (Fig. 2.1 - 4a) or by selecting *Apply action verb by rule*s from the *Apply action verbs* button's drop-down menu (Fig. 2.1 - 4b). The corresponding dialog window is shown in Fig. 2.5.

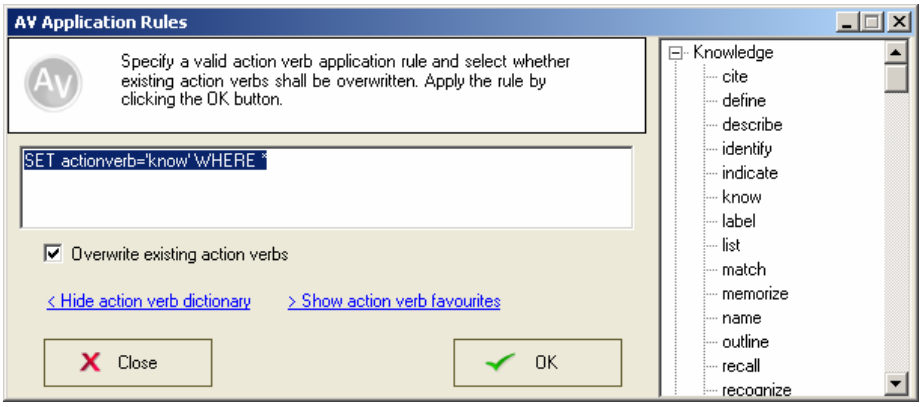


*Figure 2.5.* Action verb application rules dialog window.
On the right the action verb dictionary is displayed.

**SkOwl** offers full scripting capabilities based on SQL syntax; the reason for using this syntax is that it is very common and information on the syntax rules can easily be found on the Internet. To apply action verbs to specific skills, state the corresponding rule in the dialog window and click *OK*. In addition, you may choose to overwrite existing action verb assignments by a new rule (using the checkbox below the text area).

Each rule starts with the definition of an action verb `SET actionverb='…'`. In addition, you can establish rules by specifying to which skills the given action verb

shall be applied (term `WHERE`). You can address three parts of a skill, its object (term `obj`), subject (term `subj`), and/or its relation (term `rel`). Each addressee can be defined using an equals-statement (`=`) or a like-statement (`LIKE`). The syntax follows the rules of SQL. Please note that statements like `DELETE`, `INSERT`, `UPDATE`, or `DROP` cannot be performed! All addressees and definitions can be freely combined. Examples rules are:

- *SET actionverb='know' WHERE **
- *SET actionverb='cite' WHERE obj = 'House'*
- *SET actionverb='apply' WHERE subj LIKE '%triangle%'*
- *SET actionverb='apply' WHERE subj LIKE '%triangle%' AND rel = 'has_side'*
- *SET actionverb='apply' WHERE subj LIKE '%triangle%' OR subj LIKE '%hypo%'*
- *SET actionverb='apply' WHERE (subj LIKE '%triangle%' AND obj LIKE '%circle%') OR rel = 'has_side'*

By subsequently using different rules action verbs can be applied to all skills. In order to refer to a pre-defined set of action verbs, **SkOwl** offers an action verb dictionary, which can be displayed by clicking *Show action verb dictionary* (Fig. 2.5). By double clicking on an action verb from the dictionary, the selected action verb is automatically entered into the rule statement. It is not mandatory to use the action verbs from the dictionary, one can specify and term as action verb. However, the use of un-defined action verbs can compromise the establishment of prerequisite relations among skills. In order to allow the user to define additional action verbs and in order to easily refer to the most important action verbs, **SkOwl** offers a favorites catalogue. This catalogue can be displayed by clicking *Show action verb favorites*.

By clicking the *OK* button the rule is applied to the available skills (indicated by the purple arrow icon in the skill tree).

- *Using the rule builder*

For users who are not familiar with SQL syntax, **SkOwl** offers a rule builder to facilitate the creation of rules by providing the user with a rule form. To open the rule builder click the drop-down menu of the *Apply action verbs* button (Fig. 2.1 - 4b) and select *Rule Builder*. The rule builder window appears (Fig. 2.6).

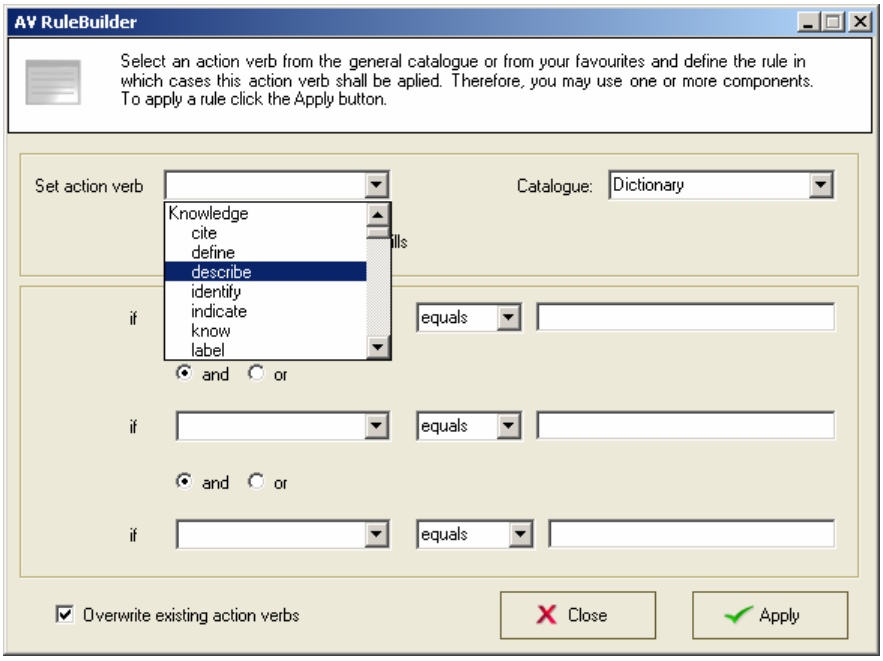


*Figure 2.6.* Action verb rule builder

To specify a rule first select an action verb from the drop-down list. This list either displays **SkOwl**'s action verb dictionary or the favorites catalogue. Which list is displayed can be selected in the *Catalogue* drop-down menu to the right. Please note that action verb categories (e.g., *Knowledge*) cannot be selected. In a second step you may select whether to set the selected action for all available skills. In this case no further entries and selections must be made and you can click the *OK* button. The selected action verb is assigned to all available skills.

In addition, you may specify up to three rule properties and combine them. To do that select and addressee (i.e., *subj, obj, rel*) from the "*if*"-drop-down menu, select a comparison method (i.e., *equals, contains*) and specify a criterion. As an example, you might select the action verb "know", you might select the addressee *subj*, the comparison method *contains*, and you might enter the criterion "triangle". This rule would assign the action verb "know" to all skills which include at least one proposition that has a subject including the term "triangle". Allows to specify three

such rules and to combine them using either *and* or and inclusive *or*. Always start specifying rules from the top. When using, for example, only the last area, no rule will be applied. Finally, select whether you want to overwrite existing action verbs with the new rule in the checkbox at the bottom of the window and click *OK*. The specified rule is applied to the available skills (indicated by the purple arrow icon in the skill tree).

- *Using batch file*

  To enable a re-use of rules, rules can be stored in an external batch file and applied within **SkOwl**. The batch file as a regular ASCII text file and it may contain any number of rules as defined above. To apply such batch file click the drop-down menu of the *Apply action verbs* button (Fig. 2.1- 4b) and select *Apply rules from batch file.* An open file dialog appears allowing to select the batch file. When the file is selected you will be asked whether to overwrite existing action verbs by new rules or not. Finally the file is processed and actions verbs are applied following the specified rules to all available skills.

- *Manually add action verbs*

  In addition to the rule-based methods of applying actions verbs, **SkOwl** also provides a manual method to add an action verb to a skill. To do that, select a skill in the skill tree. Open the context menu by a right click and select *Add action verb* from the menu. An empty action verb node is added to the selected skill, named "…". To change the action verb, select the node and make a single left click on it. When the node has switched to text mode, you may enter the desired action verb. Please note that you can specify an arbitrary action verb, however, if you use one which is not recognized by **SkOwl**'s action verb dictionary or the favorites catalogue, the establishment of prerequisite relations among the skills is compromised. All action verbs can be revised with the same method, selecting the node and single clicking to enter text mode.

  An alternative is to add an action verb from a list. To select this option, select a skill in the skill tree and open the context menu by a right click and select *Add action verb from list* from the menu. A window appears displaying either **SkOwl**'s action verb dictionary or the favorites catalogue. You can select an action verb by double clicking. The selected action verb is automatically added to the skill.

- *Deleting action verbs*

To delete an action verb or its node respectively select the action verb in the skill tree and open the right-lick context menu. Select *Delete node*. The entire node will be deleted.

#### *2.1.2.4.Add a skill manually*

In order to add a new skill, simply select the main node (named *Skills*) in the skill tree and open the context menu by a right click and select *Add skill* from the menu. A new skill is added at the end of the skill tree numbered consecutively according the specified formatting options (see section 2.1.7.). To change the skill's name, select the node and then make a single left click to enter text mode.

#### *2.1.2.5.Combining skills from multiple files*

Skills generally are based on a single source file (e.g., a proposition list or an ontology). To combine skills of multiple files, **SkOwl** offers a specific import function for skill lists. Once skills are defined on the basis of one source file you may select *Import skill list* from the skill tree's context menu. Such skill list must comply with the format described in section 2.1.2. and they must use the same delimiter as set in the main options dialog (see section 2.1.7.). The skills of the selected file are automatically appended at the end of the skill tree.

#### *2.1.2.6.Add a proposition manually*

In order to add a new proposition manually, simply select the desired skill in the skill tree and open the context menu by a right click and select *Add proposition* from the menu. A dialog window appears allowing you to specify the propositions subject, object, and relation.

#### *2.1.2.7.Changing node colors*

Because skill lists may be very long, **SkOwl** allows changing each nodes color in order to help making the skill tree more lucid. Simply select a node (skill, proposition, or action verb) in the skill tree and open the context menu by a right click and select *Change node color* from the menu. A dialog window appears allowing you to select any color. When saving a **SkOwl** project, also such color information is stored.

*2.1.2.8.Other functions*

The skill tree's context menu provides further functions. First, any node can be deleted (with the exception of the main node named *Skills*). Simply select the desired node in the skill tree and open the context menu by a right click and select *Delete node* from the menu. Second, to display all nodes with a single click select *Expand all nodes* from the context menu. Third, to hide all nodes select *Collapse all nodes* from the context menu.

Finally, all nodes of the skill tree can be renamed. To enter the text mode select a node by a single left click and by another left click a second after. The node changes to text mode and can be renamed. An exception is proposition nodes. Because propositions are defined by subject, relation, and object, such nodes cannot be simply renamed. Instead a dialog window appears allowing you to specify subject, relation, and object. When clicking *Apply* in this dialog window, the selected proposition is renamed according to those specifications.

### 2.1.3. Saving and exporting

**SkOwl** provides saving and exporting of information into following formats: **SkOwl** projects, proposition lists (txt or OWL), skill lists, and a basis / prerequisite relation (binary format or *KST tools* format).

- **SkOwl project** saves the entire information including OWL source file, propositions, skills, action verbs, and color information. To save a project click the *Save* button or use the drop-down menu of the *Save* button (Fig. 2.1 - 3b) and select *SkOwl project.*
- **Proposition lists** can be saved as ASCII files including the propositions as listed in the proposition list at the top of the *Skills* tab. Propositions contain a name or number, subject, relation, and object, divided by a delimiter. Proposition lists can be imported in other applications such as *cmap tools*. To save a proposition list use the drop-down menu of the *Save* button (Fig. 2.1 - 3b) and select *Proposition list. In the save file dialog select the file type Text (*.txt)*. In addition, the propositions can be saved in form of an OWL type I ontology. To do that, select the file type OWL ontology (*.owl) in the save file dialog.
- **Skill lists** are ASCII files including a listing of all available skills. The text file contains the name of the skills, the action verb, and the propositions, divided by a

delimiter which can be specified in the option dialog (see section 2.1.7.). To save a skill list use the drop-down menu of the *Save* button (Fig. 2.1 - 3b) and select *Skill list*.

- **Basis** (or prerequisite relation) can be stored once such matrix has been established. A relation matrix is an ASCII file including a binary coding of the prerequisites among a set of skills. This matrix can either be saved as simple binary matrix, in this case select the file type Text (*.txt) or in a form required by *KST tools* (cf. Hockemeyer, 2001). In this file format the binary matrix is prefixed with numbers of rows and columns in the matrix. For that format select the file type ASCII (*.asc) in the save file dialog. To save a matrix (if already available) use the drop-down menu of the *Save* button (Fig. 2.1 - 3b) and select *Relation Matrix.*
- **ELEKTRA ontology format** is a specific OWL ontology format (see section 1.3.). To export skills into this format a prerequisite relation among the skill must be generated first. The file contains the ontology's empty scaffold and the skills, their perquisite skills, and descriptions added. To export this format use the drop-down menu of the *Save* button (Fig. 2.1 - 3b) and select *ELEKTRA ontology model*. The current version of **SkOwl** supports version 2.0 of the ontology model.
- **Inserting into existing ELEKTRA ontologies**: To insert skills, their prerequisite relations, and descriptions, use the drop-down menu of the *Save* button (Fig. 2.1 - 3b) select *Insert into ELEKTRA ontology.* The available skill information is added to the existing ontology file.

### 2.1.4. Generating a basis / prerequisite relation

**SkOwl** supports the semi-automatic establishment of basis / prerequisite relations between skills and to transform the basis / prerequisite relation into a binary relation matrix. Once the skills are defined (with or without action verbs), a "draft" basis / prerequisite relation can be generated. By default, **SkOwl** generates a basis file. In the options dialog it is possible to specify to generate a prerequisite relation file. Basically, these contain the same information, however in comparison to a basis, a prerequisite relation matrix is rotated. **SkOwl** thus offers two functions:

- **Using the set inclusion principle and the action verb level**: The set inclusion principle (see section 1.4.) basically states that when a skill's propositions are a

subset of another skill's proposition, the first skill is a prerequisite for the second skill. In cases where skills have the same propositions, the level of the assigned action verbs determines a prerequisite relation, stating that the skill with the "lower" action verb level (e.g., *knowledge*) is a prerequisite for the skill with the "higher" action verb level (e.g., *application*). For pairs of skills that do not constitute sub-sets, no prerequisite relations are suggested, independent from the corresponding action verb levels. To use this option, click the *Generate the relation matrix* button or by selecting *Make relation matrix* from this button's drop-down menu (Fig. 2.1 - 3a and 3b).

- **Using specific rules**: Basically, this method is based on the same principles as the first method. However, **SkOwl** acknowledges that there may be specific cases, when aforementioned principles cannot be applied. Example: One skill's propositions may be a subset of a second skill's propositions, however, the number of propositions of the second skill is only slightly higher then that of the first one. In contrast, the action verb level of the first skill might be substantially "higher" than that of the second one. In this particular case, the set inclusion principle might not be applied. Instead this principle might be invert, meaning that in this case the first skill is seen as a prerequisite for the second one.

  To address such complex relationships, **SkOwl** offers a relation rule builder, which can be opened by selecting *Make relation matrix using rules* from the *Make relation matrix* button's drop-down menu (Fig. 2.1 - 3b) or by clicking the Start rule builder button in the toolbar (Fig. 2.1 - 6).

  The rule builder dialog window is shown in Fig. 2.7; rules can be defined in following way. First specify whether the set inclusion principle shall not be applied or if it shall even be inverted. Subsequently define for which skills the specified rule shall be applied. **SkOwl** allows specifying a key string and a comparison method (*equals, contains*). Moreover, it can be defined which difference between action verb levels must exist and also which difference must exist between the skill's numbers of propositions. By clicking the *Add* button a specified rule is added to the set of rules applied for generating the prerequisite relations among the skills. In turn, by clicking the *Del* button, a selected rule can be removed from the list. A given set of rules can be saved and, of course, be opened again using the corresponding buttons. In order to

clear the entire set of rules click the *Clear rules* button. To apply a set of rules, click the *Update* button. When generating the prerequisite relations between skills, the rules are applied in the same order as listed in the rules list (Fig. 2.7). As a consequence, later rules might overwrite rules applied earlier. By using the arrow buttons, the order of rules can be influenced. Still, it is necessary to carefully define rules and put them in an appropriate order!

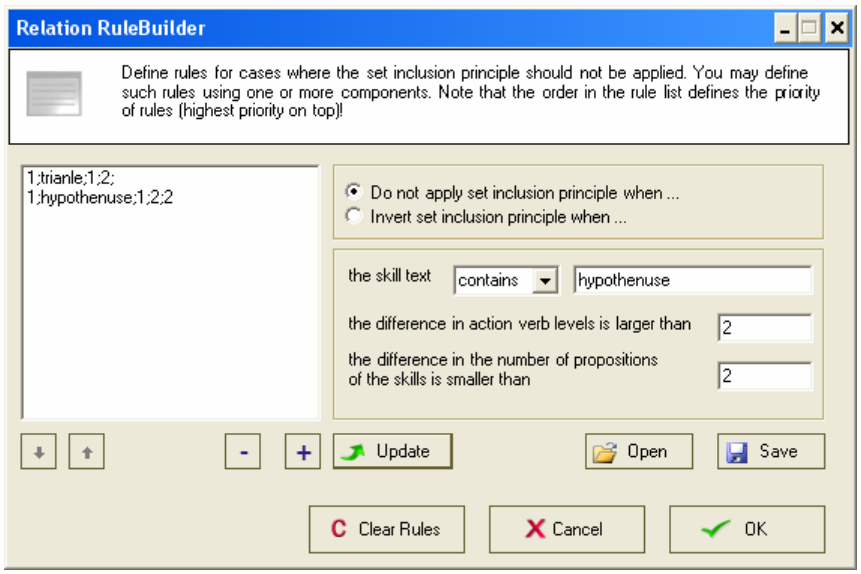


Figure 2.7. Relation rule builder

### 2.1.5. Print skill list

Using the *Print* button (Fig. 2.1 - 8), the available list of skills can be printed. When clicking this button a printer dialog appears allowing the selection of a printer and corresponding printing options. Equally to the export of the skill list, the skill tree's skills are printed in form of skill name, action verb, and proposition(s).

*Figure 2.8.* Action verbs favorites catalogue editor

### 2.1.6. Editing action verb favorites catalogue

As mentioned in previous sections, **SkOwl** allows establishing a catalog of favorite action verbs. To start the maintenance dialog window (Fig. 2.8) click the *Edit favorites catalogue* button (Fig. 2.1 - 9).

To edit the favorites catalogue, first select an action verb level from the drop-down menu. The available actions verbs of the selected level are listed automatically. To add a new action verb, enter it in the text box below the action verb list and click the *Add* button. In order to delete an existing action verb, select it in the list by a left click and click the *Del* button. In order to clear the entire list, click the *Clear* button.

### 2.1.7. Application options

**SkOwl** offers an option dialog in order to make specific application settings (Fig. 2.9). The options dialog includes two pages, one for the main application and one for the relation visualizer module.

On the main application page following settings can be made: (a) To select whether to display the splash screen at application start, (b) to select whether to use the action verb level when creating the relation matrix (or the set inclusion principle only), (c) to select whether to generate a basis file or a prerequisite relation file, (d) to specify delimiters for exporting proposition and skill lists (this may be any valid Unicode character or “tab” for a tabulator),

and (e) the format for automatic skill enumeration. The latter consists of a prefix (e.g., “S” or “Skill”) and a number format (e.g., “000” for a three-digit format such as “012”).

On the relation visualizer page following settings can be made: (a) To select the color for the background, the skills, and the graph’s lines (a color selection dialog window is displayed when clicking on the colored boxes), (b) to define a font style for skills (a font selection dialog window is displayed when clicking on he font box), (c) to define a pen width for the graph’s lines, (d) the maximum length (in characters) for the skill labels, as well as (e) X and Y axis scale values for drawing prerequisite relations as well as skill/knowledge structures. The latter values determine the drawing size (in terms of width and height) for the skills/skill states.

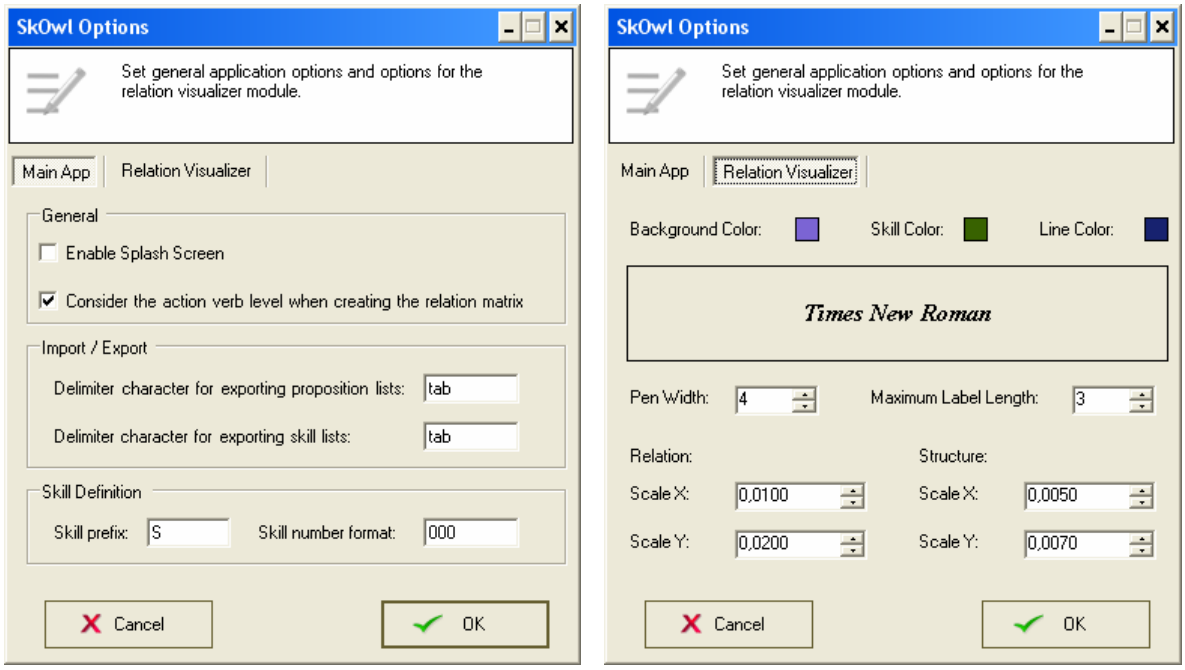


*Figure 2.9.* Main application options dialog window

### 2.1.8. Help and additional function

**SkOwl** provides certain help functions and additional application information. By clicking the *Show manual* button (Fig. 2.1 -11a) or selecting *Show manual (pdf)* from its drop-down menu (Fig. 2.1 -11b), this manual can be displayed at runtime. To display this manual in PDF format, the default browser is opened. Please note that this function requires a default web browser (e.g., the Microsoft Internet Explorer) and Adobe Acrobat (Reader) installed. In addition, the drop-down menu offer following functions: (a) To display the license agreement,

(b) to open the website of the Cognitive Science Section (CSS), Department of Psychology, University of Graz, Austria., and (c) display an *about window*.

## 2.2 SkOwl's relation visualizer

**SkOwl** provides a relation visualizer as a major component (Fig.2.10). This module draws a relation matrix or in other terms a basis in form of a vector-oriented Hasse diagram. The vector-oriented approach allows manipulating the original relation matrix. Once a basis has been generated this basis can be directly drawn with the visualizer module. To start the module click the *Start visual relation builder* button (Fig. 2.1 -7a) or select *Visualize relation (with labels)* from this buttons drop-down menu (Fig. 1 - 7b). The relation visualizer window appears and the available basis is drawn according to the given settings (see section 2.1.7.). When using this function, the skills are drawn using the skill's labels (e.g., S001). To draw the skills using only numbers, select *Visualize relation (without labels)* from the drop-down menu.

The relation visualizer modules is based on a third-party ActiveX control (see http://www.script-debugger.com), which is free for non-commercial use. Still, in the present version of **SkOwl** the graphics area is marked with the term "Unregistered Copy".

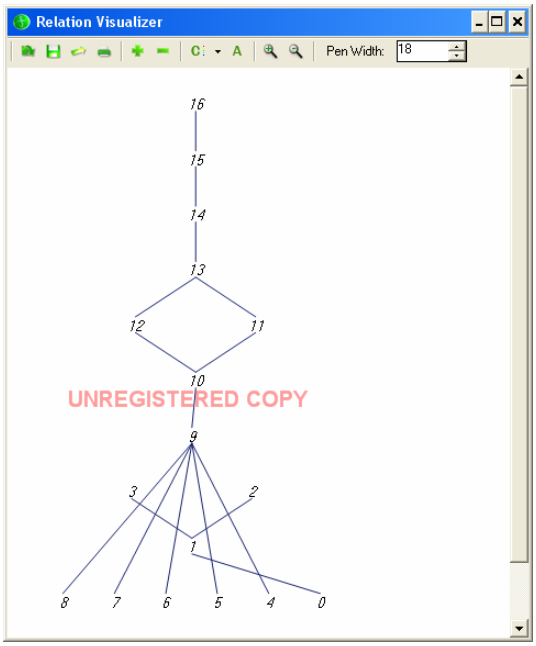


*Figure 2.10.* Relation visualizer module

### 2.2.1 Starting the relation visualizer module

The relation visualizer (Fig. 2.10) can be opened directly drawing a relation matrix generated with **SkOwl**. Simply click the *Start visual relation builder* button (Fig.1 -7a) or select *Visualize relation (with labels)* or *Visualize relation (without labels)* from the drop-down menu (Fig. 1 - 7b). To start the relation visualizer without drawing an available relation matrix, select *Start relation visualizer* from the drop-down menu.

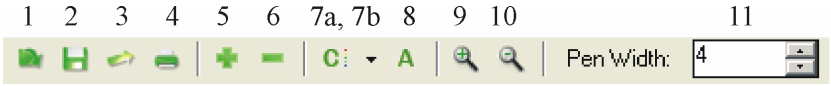


*Figure 2.11*. Toolbar of the relation visualizer module

### 2.2.2 Open source files

The relation visualizer module supports following file formats:

| | |
|---|---|
| ***SkOwl** relation images (*.sri)* | This is **SkOwl**'s internal file format allowing saving a generated graph including background, text, and line color, as well as size and font style. |
| *Bass files (*.txt)* | Basis files are ASCII file including a binary representation of a prerequisite relation in form of a matrix. |
| *KST tools basis files (*.txt)* | Such files are basis files; however they include the number of rows and columns at the beginning of the file. |
| *Space file (*.txt)* | Space files are ASCII files including a binary representation of a skill or knowledge structure in form of a item x state matrix. Such files may include up to several millions of skill or knowledge states. |

To open a source file click to *Open source file* button (Fig. 2.11 - 1) and select the desired file type in the appearing open file dialog window. **SkOwl** interprets the file and draws the file's information in form of a Hasse diagram. When opening files other than **SkOwl** relation images (*.sri), you will be asked whether to open a label file also. Label files contain information about the labels/names of each item (column) of relation, basis, or structure files. Label files are ASCII files including one label in each line, ordered as the order of columns in the relation, basis, or structure file. When answering yes, another open file dialog window

appears. Select the associated label file and click OK. When answering no, the Hasse diagram will be drawn without labels. Skills are numbered consecutively according their order in the source file.

**A note on drawing space files:** Space file may be very extensive files including thousands or even millions of states. To draw such spaces, on the one hand, is a highly resource consuming process which may take a significant amount of time. On the other hand, very large space would result in lines containing thousands of states. Such lines cannot be displayed appropriately. For example to (visibly) print a line containing 10.000 states would require to have 300 (!) A4 pages in landscape format next to each other. Thus, it is recommended to draw spaces including up to 100 or 200 states per line.

When drawing as space file, it is possible to determine the number of possible learning paths for a given knowledge/skill structure. To enable this option check the corresponding box in the main options dialog window (see section 2.1.7).

**Please note that when drawing a space file, all manipulation functions (e.g., moving nodes, deleting nodes, or lines, changing colors) do not work!**

### 2.2.3 Saving basis files and images

The relation visualizer module allows saving either the drawn images or the corresponding basis file. Following file types are supported:

| | |
|---|---|
| ***SkOwl** relation images (*.sri)* | This is **SkOwl**'s internal file format allowing saving a generated graph including background, text, and line color, as well as size and font style. |
| *Bass files (*.txt)* | Basis files are ASCII file including a binary representation of a prerequisite relation in form of a matrix. |
| *KST tools basis files (*.txt)* | Such files are basis files, however they include the number of rows and columns at the beginning of the file. |
| *Bitmap files (*.bmp)* | Uncompressed image format |
| *JPEG files (*.jpg)* | Compressed image format |
| *ELEKTRA ontology format (*.owl)* | The ELEKTRA ontology model is a specific ontology format based on OWL DL. |

| | |
|---|---|
| *Insert into ELEKTRA ontology (*.owl)* | Insert the given information into an existing ELEKTRA format ontology file. |

To save file click to *Save file* button (Fig. 2.11 - 2) and select the desired file type in the appearing save file dialog window. For saving a relation to OWL ELEKTRA format click the *Export OWL* button (Fig. 2.11 - 3). In the file type menu you can select either to create a new file (selecting *ELEKTRA ontology model*; providing the scaffold of the ELEKTRA ontology model with added skills) or to insert the skills and their prerequisite relations to an existing file (selecting *Insert into ELEKTRA ontology*).

### 2.2.4 Printing images

**SkOwl** allows printing an available image / Hasse diagram. Simply click the *Print* button (Fig. 2.11 - 4). A dialog window appears within which the print settings can be made.

### 2.2.5 Manipulating the graph

A major strength of the relation visualizer module is that the visualization is based on vector graphics, enabling the manipulation of the relation. Such manipulation can be made by means of moving skills, adding relations (lines), and by removing relations (lines).

- **Moving skills**

  To move a skill, select the skill by a mouse click (the skills is shaded grey) and move the move with pressed left mouse button. If the moved skills as one or more relations (lines), these are automatically moved with the skill. The alignment of relations (top or bottom) is adjusted automatically.

- **Adding relations (lines)**

  To add a relation (line) between two skills, select the first skill (no matter if the skills is on a lower, higher, or the same level) by a single left click. Select a second skill by a single right (!) click. Now both skills are shaded grey. To add a relation, click the *Add relation* button (Fig. 2.11 - 5). **SkOwl** automatically checks whether the desired relation is already covered by the transitivity property. As an example, when *A* is related to *B* and *B* is related to *C*, *C* is (indirectly) related to *A* by transitivity. If such transitivity property is encountered, you will be asked if you want to add the relation anyway (click yes or no in the message box).

- **Removing relations (lines)**

  To remove a relation (line), select the desired relation by a single left click. The relation is shaded grey. To remove it, click the *Delete relation* button (Fig. 2.11 - 6).

#### *2.2.5.1 Changing colors*

By default, the relation visualizer module uses the colors specified in the main options dialog (see section 2.1.7.). To use different colors for a particular image, **SkOwl** allows changing the color of single items (skills or relations) or of entire item groups.

To change the color of a single item (a skill or a relation), select the item by a single left click and click the *Change color* button (Fig. 2.11 - 7a). A color selection dialog appears.

To change the color of entire item groups select the corresponding option for the *Change color* button's drop-down menu (Fig. 2.11 - 7b). The available options are *Set background color, Set font color* (skill color), and *Set relation color*.

#### *2.2.5.2 Changing font style*

To change the font style for all skills, click the *Change font* button (Fig. 2.11 - 8) and select the desired font style in the appearing dialog window.

#### *2.2.5.3 Changing pen (line) width*

By default, the relation visualizer module uses the line width specified in the main options dialog (see section 2.1.7.). To change the line width for a specific image, use the size filed (Fig. 2.11 - 11) by clicking up or down. The line width is adjusted automatically. The minimum line width is 1 and the maximum 100.

#### *2.2.5.4 Zooming in and out*

To change the size of the image, you may zoom in and out. Simply click on the *Zoom in* or *Zoom out* button until the desired display size is reached (Fig. 2.11 - 9 and 10).

# Section 3: Architecture and Specifications

## 3.1. Architecture

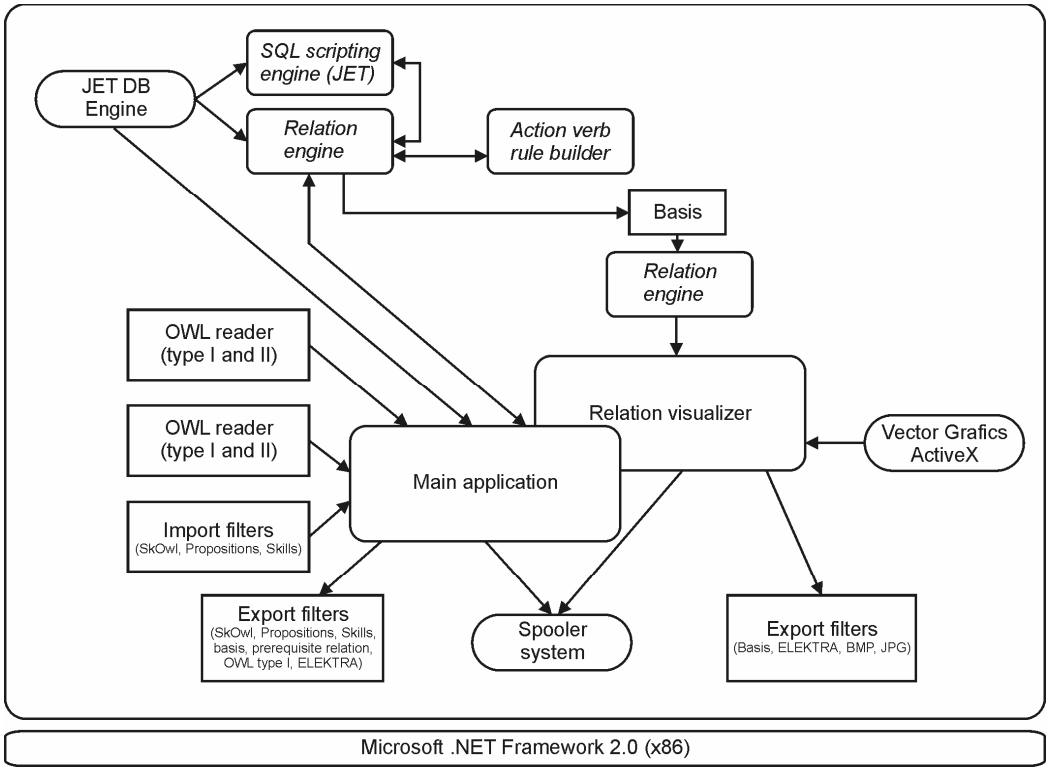


*Figure 3.1.* **SkOwl**'s architecture

**SkOwl** is developed on the basis of *VB.NET*, using *Microsoft Visual Studio 2003*. **SkOwl** is based on the *Microsoft .NET framework 2.0*. It is designed as single document interface (SDI) based on two main modules, the main application window and the relation visualizer module. In addition, **SkOwl** implements following external components (for further details please refer to **SkOwl**'s source code):

- **Vector Graphics ActiveX**

  This third-party component is used for the vector-oriented drawing of basis and even structure files in the relation visualizer module. Vector Graphics ActiveX (www.script-debugger.com) is professional graphics solution for creating business and

technical drawings. It provides a variety methods, properties, and events and is fully object oriented. Top features are:

- Easy in use
- Hardware acceleration
- High-speed or high-quality rendering with bicubic filtration and antialiasing
- Export drawings to BMP, JPG, PNG and EMF formats
- Ability to establish connections between shapes and external data
- Changing all attributes in design and runtime modes
- All mouse events, mouse wheel, scroll and zoom
- Bezier curves and Cardinal splines
- Shapes with holes
- Filling shapes with color, gradient, sketch or texture
- Transparent contours and brushes
- Patterns and shapes libraries
- Flexible Printing

Vector Graphics ActiveX is free for noncommercial use; however, in the current and previous versions of **SkOwl**, the text "Unregistered copy" is display in the graphic area of the relation visualizer module.

- **Microsoft Jet 4.0 database engine**

  The *JET* (Joint Engine Technology) engine allows manipulating relational databases and is part of the *Relational Database Management System* (RDBMS) and provides support for security, referential integrity, transaction processing, indexing, record, page locking, and data replication. *JET* enables SQL queries, store character data in Unicode format, create database views and allowed bi-directional replication with *Microsoft SQL Server*. JET includes three modules, the Native Jet ISAM Driver, a dynamic link library (DLL) that could directly manipulate Microsoft Access database files (MDB), which was a modified form of an Indexed Sequential Access Method (ISAM) database, and the ISAM Drivers, DLLs that allowed access to ISAM databases, among them being Xbase, Paradox, Btrieve and FoxPro files.

  **SkOwl** uses the *JET* engine for retrieving action verbs from a *Microsoft Access* databases and for provide full SQL scripting support in the relation rule builder.

- **Microsoft .NET Framework Version 2.0 Redistributable Package (x86)**
  Applications and controls written for the Microsoft .NET Framework require the .NET Framework to be installed on the computer where the application or control runs. The .NET Framework version 2.0 provides one redistributable installer, dotnetfx.exe, which contains the common language runtime and .NET Framework components necessary to run .NET Framework applications.

## 3.2. System Requirements

- **Supported Operating Systems**
  Windows 2000 Service Pack 3; Windows 98; Windows 98 Second Edition; Windows ME; Windows Server 2003; Windows Vista Business; Windows Vista Business 64-bit edition; Windows Vista Enterprise; Windows Vista Enterprise 64-bit edition; Windows Vista Home Basic; Windows Vista Home Basic 64-bit edition; Windows Vista Home Premium; Windows Vista Home Premium 64-bit edition; Windows Vista Starter; Windows Vista Ultimate; Windows Vista Ultimate 64-bit edition; Windows XP Service Pack 2

- **Required Software**
  - Windows Installer 3.0 (except for Windows 98/ME, which require Windows Installer 2.0 or later). Windows Installer 3.1 or later is recommended
  - IE 5.01 or later: You must also be running Microsoft Internet Explorer 5.01 or later for all installations of the .NET Framework
  - Adobe Acrobat or Acrobat Reader, version 4 or higher

- **Required Hardware**
  - Intel Pentium or an equivalent processor with a core frequency of 166 MHz or higher
  - 128 MB RAM
  - 280 MB free disk space for the Microsoft .NET framework (x86)
  - 10 MB free disk space for the **SkOwl** installation

- **SkOwl** has been successfully tested on following configurations
  - Configuration 1
    - *Desktop PC (Dell Optiplex GX620)*
    - *Intel Pentium D processor, 2.8 GHz*
    - *1 GB RAM*
    - *Microsoft Windows XP Service Pack 2*
    - *Microsoft .NET Framework redistributable package 2.0 (x86)*
    - *Vector Graphics ActiveX installed*
    - *Microsoft Internet Explorer 7 installed*
    - *Adobe Acrobat 5.0 installed*
  - Configuration 2
    - *Notebook PC (Sony Vaio VGN-S5)*
    - *Intel Pentium M processor, 1.73 GHz*
    - *512 MB RAM*
    - *Microsoft Windows XP Service Pack 2*
    - *Microsoft .NET Framework redistributable package 2.0 (x86)*
    - *Vector Graphics ActiveX installed*
    - *Microsoft Internet Explorer 6 installed*
    - *Adobe Acrobat 5.0 installed*

## 3.3. Action verb dictionary

**SkOwl**'s action verb dictionary in the current version includes following action verbs:

| | |
|---|---|
| *Knowledge* | know, cite, define, describe, identify, indicate, know, label, list, match, memorize, name, outline, recall, recognize, record, relate, repeat, reproduce, select, state, underline |
| *Comprehension* | arrange, classify, convert, defend, describe, diagram, discuss, distinguish, estimate, explain, extend, generalize, give examples, infer, locate, outline, paraphrase, predict, report, restate, review, suggest, summarize, translate |

| | |
|---|---|
| *Application* | apply, change, compute, construct, demonstrate, discover, dramatize, employ, illustrate, interpret, investigate, manipulate, modify, operate, organize, practice, predict, prepare, produce, schedule, shop, sketch, solve, translate, use |
| *Analysis* | analyze, appraise, break down, calculate, categorize, compare, contrast, criticize, debate, determine, diagram, differentiate, discriminate, distinguish, examine, experiment, identify, illustrate, infer, inspect, inventory, outline, question, relate, select, solve, test |
| *Synthesis* | arrange, assemble, categorize, collect, combine, compile, compose, construct, create, design, devise, explain, formulate, generate, manage, modify, organize, perform, plan, prepare, produce, propose, rearrange, reconstruct, relate, reorganize |
| *Evaluation* | appraise, assess, choose, compare, conclude, contrast, criticize, decide, discriminate, estimate, evaluate, grade, interpret, judge, justify, measure, rate, relate, revise, score, select, summarize, support, value |

## 3.4. Error codes

**SkOwl** includes certain error handling and identifications routines. Generally, the user is provided with a message when an error occurs, which includes an error code to identify an error, and the corresponding *sub* or *function* is closed. The logic behind error code format is as follows: Error codes are displayed as five-digit numbers. The first two digits denote the .NET framework *system.component.windows.forms* model entity of the application (short, the *forms* component). The following three digits denote the *sub* or *function* within which the error occurred. Table 3.1 lists the error codes used in **SkOwl** Generally, there are two types of error messages, warnings and errors (stated in the messages' titles). Warnings indicate errors which occurred by an incorrect use of the application and which do not fatally influence the execution of the application or the integrity of a project. Errors indicate fatal errors which

seriously compromise the integrity of the application or a project respectively. Due to error handling procedures in most cases this does not lead to a closing of the entire application. Reasons for errors are usually deleting or displacing (temporary) files, e.g., the original OWL file, or the unavailability of specific system components (e.g., COM components or components of the Microsoft .NET framework).

**Example:** A message may say "An unexpected error occurred! Error code: 01002.". This message states that an error occurred in the *forms* component *frmMain* in the *sub* that handles the *Button4.Click* event.

Table 3.1. *Error codes in **SkOwl***

| Error Code | Component | (Sub-) Function | Error Description |
|---|---|---|---|
| 01001 | frmMain | frmMain_Load | Denotes an error that occurred at the application's start procedure caused by removing or displacing the application's main configuration file (scowl.ini). |
| 01002 | frmMain | Button4_Click | Denotes an error that occurred while saving a project. Basically, this might refer to *ini*-functions or the reading from ActiveX *system.component. windows.forms* controls. |
| 01003 | frmMain | Button5_Click | Denotes an error that occurred while loading a saved project. Basically, this might refer to *ini*-functions or the writing to ActiveX *system.component. windows.forms* controls. |
| 01004 | frmMain | OpenOWLFile | Denotes an error that occurred while reading an OWL file in the course of loading a saved project. Reasons may be caused by removing or displacing the original OWL file or by an invalid or corrupted OWL file. |
| 01005 | frmMain | FillDB | Denotes an error while clearing a temporary Microsoft Access database. |
| 01006 | frmMain | FillDB | Denotes an error in writing proposition data to the temporary database table. |
| 01007 | frmMain | ApplyActionVerbs | Denotes an error while query temporary data in rule-based application of action verbs. |
| 01008 | frmMain | oTv_DragDrop | Denotes an error that occurred while attempting to drag and drop items within the skilltree or from the proposition list. |
| 01009 | frmMain | MakeBasis | Denotes an error that occurred while establishing a list of unique skills from the *SkillTree1* control. |
| 01010 | frmMain | StoreInDB | Denotes an error that occurred while writing the unique skill descriptions (by propositions and action verbs) to a temporary database. |
| 01011 | frmMain | clearSets | Denotes an error while clearing a temporary Microsoft Access database. |
| 01012 | frmMain | GetAVLevel1 | Denotes an error that occurred while determining the level id of action verbs. |

| | | | |
|---|---|---|---|
| 01013 | frmMain | GetAVLevel2 | Denotes an error that occurred while determining the level id of action verbs. |
| 01014 | frmMain | MakeBasis | Denotes an error that occurred while preparing the relational matrix (i.e., the basis). |
| 01015 | frmMain | MakeBasis | Denotes an error that occurred while determining dependencies based on the propositions as well as action verbs in the relational matrix (i.e., the basis). |
| 01016 | frmMain | OpenPropositions | Denotes an error that occurred while opening a proposition list file. This error may be caused by an invalid or corrupted proposition list file or by specifying an incorrect delimiter (tab, semicolon, comma, or space) in the open dialog. |
| 01017 | frmMain | ExpPropositions | Denotes an error that occurred while exporting a proposition list to an ASCII file. |
| 01018 | frmMain | ExpSkills | Denotes an error that occurred while exporting a skill list to an ASCII file. |
| 01019 | frmMain | ApplyFromBatch | Denotes an error that occurred while assigning action verbs to the skills by using a rule batch file. |
| 01020 | frmMain | ImportSkills | Denotes an error that occurred while importing a skill list file. |
| 01021 | frmMain | ExpELEKTRA | Denotes an error that occurred while exporting skills and their prerequisite relations to the ELEKTRA ontology format. |
| 01022 | frmMain | InsELEKTRA | Denotes an error that occurred while inserting skills and their prerequisite relations to an existing ELEKTRA ontology format file. |
| 02001 | frmRuleBuilder | LoadActionVerbs | Denotes an error that occurred while loading the action verb catalogue. This error may occur for example when the catalogue-file (actionverbs.mdb) has been removed or displaced. |
| 02002 | frmRuleBuilder | frmRUleBuilder2_Load | Denotes an unknown error while loading the AV RuleBuilder. A possible reason is that the SkOwl configuration file (skowl.ini) has been removed or corrupted. A possible solution to this error is to re-install SkOwl. |
| 02003 | frmRuleBuilder | Button1_Click | Denotes an unknown error while trying to apply the rule to the generation of action verbs. |
| 03001 | frmFavEdit | LoadVerbs | Denotes an error that occurred while loading the action verb catalogue. This error may occur for example when the catalogue-file (actionverbs.mdb) has been removed or displaced. |
| 03002 | frmFavEdit | Button2_Click | Denotes an error that occurred while loading the action verb catalogue. This error may occur for example when the catalogue-file (actionverbs.mdb) has been removed or displaced. |
| 03003 | frmFavEdit | UpdateAdd | Denotes an error that occurred while adding a new action verb to the favourites catalogue. |
| 03004 | frmFavEdit | UpdateDel | Denotes an error that occurred while deleting an existing action verb from the favourites catalogue. |
| 03005 | frmFavEdit | Button3_Click | Denotes an error that occurred while clearing the favourites catalogue. |
| 04001 | frmRuleBuilder2 | Button7_Click | Denotes a not closer specified error that occurred by opening an invalid rule collection. |

| | | | |
|---|---|---|---|
| 04002 | frmRuleBuilder2 | Button7_Click | Denotes an error that occurred by opening an invalid rule collection because the number of rule components was invalid. |
| 04003 | frmRuleBuilder2 | frmRUleBuilder2_Load | Denotes an unknown error while loading the relation rule builder. |
| 05001 | modGlobal | PopulateClassTree_OWL1 | Denotes an error that occurred while parse an OWL type I ontology and populating the class and instance information in a treeview control. A reason is likely that the specified OWL file is not a valid type I file. |
| 05002 | modGlobal | GetInstances_OWL1 | Denotes an error that occurred while retrieving propositions from an OWL type I ontology. |
| 05003 | modGlobal | GetOWL2 | Denotes an error that occurred while parsing an OWL type II ontology. A reason is likely that the specified OWL file includes unsupported properties. |
| 05004 | modGlobal | PopulateSkillTree | Denotes an unexpected error while populating the skill treeview. |
| 06001 | frmSQL | LoadAV() | Denotes an error that occurred while loading the action verb catalogue. This error may occur for example when the catalogue-file (actionverbs.mdb) has been removed or displaced. |
| 07001 | frmOptions | frmOptions_Load | Denotes an unknown error that occurred while loading the options dialog. A possible reason for this error is that the SkOwl configuration file (skowl.ini) has been removed or corrupted. A possible solution to this error is to re-install SkOwl. |
| 07001 | frmOptions | Button1_Click | Denotes an unknown error that occurred while saving settings. A possible reason for this error is that the SkOwl configuration file (skowl.ini) has been removed or corrupted. A possible solution to this error is to re-install SkOwl. |
| 08001 | frmAVList | LoadAV() | Denotes an error that occurred while loading the action verb catalogue. This error may occur for example when the catalogue-file (actionverbs.mdb) has been removed or displaced. |
| 09001 | frmVis1 | ReadBasis | Denotes an error that occurred while reading a basis file. A possible reason is that the basis file has an invalid format or is corrupted. Please note that in contrast to the applications *constr* or *di* (Hockemeyer, 2001), SkOwl does not allow the file starting with a line for the number of items and a line for the number of states. |
| 09002 | frmVis1 | GetSupersets | Denotes an unexpected while determining supersets of each basis state. A possible reason is that the basis file has an invalid format or is corrupted. |
| 09003 | frmVis1 | GetScale | Denotes an unknown error that occurred while initializing the drawing area. |
| 09004 | frmVis1 | DrawSkills | Denotes an error that occurred while determining and drawing the skills. |
| 09005 | frmVis1 | DrawLines | Denotes an error that occurred while drawing the relations (lines) between skills. |
| 09006 | frmVis1 | DLBTMove | Denotes an error that occurred while moving skills. |
| 09007 | frmVis1 | AddLine | Denotes an unknown error that occurred while adding a new relation (line) to the relation. |

| | | | |
|---|---|---|---|
| 09008 | frmVis1 | DelLine | Denotes an unknown error that occurred while deleting a relation (line), |
| 09009 | frmVis1 | MakeBasis | Denotes an error that occurred while deriving the basis file from the drawing. |
| 09010 | frmVis1 | SaveSRI | Denotes an error that occurred while saving a SkOwl relation image file (*.sri). |
| 09011 | frmVis1 | LoadSRI | Denotes an error that occurred while loading a SkOwl relation image file (*.sri). A possible reason for this error is opening an invalid file. |
| 09012 | frmVis1 | SaveFile | Denotes an error that occurred while saving a SkOwl relation image file (*.sri). A possible reason is that a file which is write-protected shall be overwritten. |
| 09013 | frmVis1 | ReadLabels | Denotes an error that occurred while reading a skill label file. |
| 09014 | frmVis1 | GetSpaceScale | Denotes an error that occurred while setting the scale for the drawing area based on the number of items, the maximum number of states per level, and the settings for the width and height for each item. |
| 09015 | frmVis1 | DrawStates | Denotes an error that occurred while drawing the knowledge structure or skill structure respectively. |
| 09016 | frmVis1 | DrawSpace | Denotes an error that occurred while preparing data to draw the knowledge structure or skill structure respectively. |
| 09017 | frmVis1 | ExpELEKTRA | Denotes an error that occurred while exporting skills and their prerequisite relations to the ELEKTRA ontology format. |
| 0918 | frmVis1 | InsELEKTRA | Denotes an error that occurred while inserting skills and their prerequisite relations to an existing ELEKTRA ontology format file. |

# Section 4: License agreements

## 4.1. SkOwl

**SkOwl** and its source code are distributed under GNU General Public License
Copyright (C) 1989, 1991 Free Software Foundation, Inc.
51 Franklin Street, Fifth Floor, Boston, MA  02110-1301, USA

Excepted from these terms and conditions is the third-party component Vector Graphics ActiveX. See section 4.2. for separate license agreements.

**Preamble**

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Lesser General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

**TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION**

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in objects code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of

this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.
This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those

countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

**NO WARRANTY**

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE

LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

## 4.2. Vector Graphics Active X

Vector Graphics ActiveX is distributed under following terms and conditions:

**NO WARRANTY**

VECTOR GRAPHICS ACTIVEX IS SOLD "AS IS" AND WITHOUT ANY WARRANTY AS TO MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE OR ANY OTHER WARRANTIES EITHER EXPRESSED OR IMPLIED. THE AUTHOR WILL NOT BE LIABLE FOR DATA LOSS, DAMAGES, LOSS OF PROFITS OR ANY OTHER KIND OF LOSS WHILE USING OR MISUSING THIS SOFTWARE.

**Evaluation and Registration**

Vector Graphics ActiveX is free for using as a part of any application that embeds it. If you wish to embed Vector Graphics ActiveX into your applications or distribute it as a part of your application you must buy Developer License (one Developer License per one application that embeds ActiveX).

**Distribution of the evaluation version**

You may copy the evaluation version of this software and documentation as you wish, and give exact copies of the original evaluation version to anyone, and distribute the evaluation version of the software and documentation in its unmodified form via electronic means. But you should not charge or requesting donations for any such copies however made and from distributing the software and/or documentation with other products without the author's written permission.

**Registered version**

Developer License allows you to embed Vector Graphics ActiveX into your application and then distribute it as a part of your software without any restrictions.

# Section 5: References


Albert, D., & Held, T. (1999). Component based knowledge spaces in problem solving and inductive Reasoning. In D. Albert & J. Lukas (Eds.), *Knowledge Spaces: Theories, empirical research, applications* (pp. 15-40). Mahwah, NJ: Lawrence Erlbaum.

Anderson, J.R. (1996). A simple theory of complex cognition. *American Psychologist*, 51, 355-365.

Anderson, L.W., & Krathwohl, D.R. (2001). *A taxonomy for learning, teaching, and assessing: A revision of Bloom's taxonomy of educational objectives*. New York: Longman.

Antoniou, G., & van Harmelen, F. (2004). Web ontology language: OWL. In S. Staab & R. Studer (Eds.), *Handbook on ontologies* (pp. 67-92), Berlin: Springer-Verlag.

Bloom, B.S. (Ed.) (1956). *Taxonomy of educational objectives: The classification of educational goals. New* York: Longmans Green.

Branko Neto, W. C., & Gauthier, F.A. (2006). Sharing and reusing information on web-based learning. Workshop on *Applications of Semantic Web Technologies for e-Learning,* June 21-23, 2006.

Gómez-Pérez, A., & Corcho, O. (2002). Ontology languages for the semantic web. *IEEE Intelligent Systems*, 17 (1), 54-60.

Gruber, T.R. (1993). A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5 (2), 199-220.

Heller, J., Steiner, C.M., Hockemeyer, C., & Albert, D. (2006). Competence-based knowledge structures for personalized learning. *International Journal on E-Learning*, 5, 75-88.

Hockemeyer, C. (2001). *KST Tools User Manual* (Unpublished Technical Report). Institut für Psychologie Karl–Franzens–Universität Graz, Austria.

Kickmeier-Rust, M.D., & Albert, D. (in press). *The ELEKTRA ontology model: Towards a learner-centered approach to resource description.*

Mizoguchi, R. & Bourdeau, J. (2000). Using ontological engineering to overcome common AI-ED problems. *International Journal of AI in Eduction*, 11, 107-121.

Sampson, D., Karagiannidis, C., & Cardinali, F. (2002). An architecture for web-based e-learning promoting re-usable adaptive educational e-content. *Educational Technology & Society,* 5 (4), 27-37.

Steiner, C.M., & Albert, D. (in press). *Personalising learning through prerequisite structures derived from concept maps.*